
Machine Learning

Topic 5: Linear Discriminants

Bryan Pardo, EECS 349 Machine Learning, 2013

Thanks to Mark Cartwright for his extensive contributions to these slides
Thanks to Alpaydin, Bishop, and Duda/Hart/Stork for images and ideas

General Classification Learning Task

There is a set of possible examples $X = \{\vec{x}_1, \dots, \vec{x}_n\}$

Each example is an k -tuple of attribute values

$$\vec{x}_1 = \langle a_1, \dots, a_k \rangle$$

There is a target function that maps X onto some finite set Y

$$f : X \rightarrow Y$$

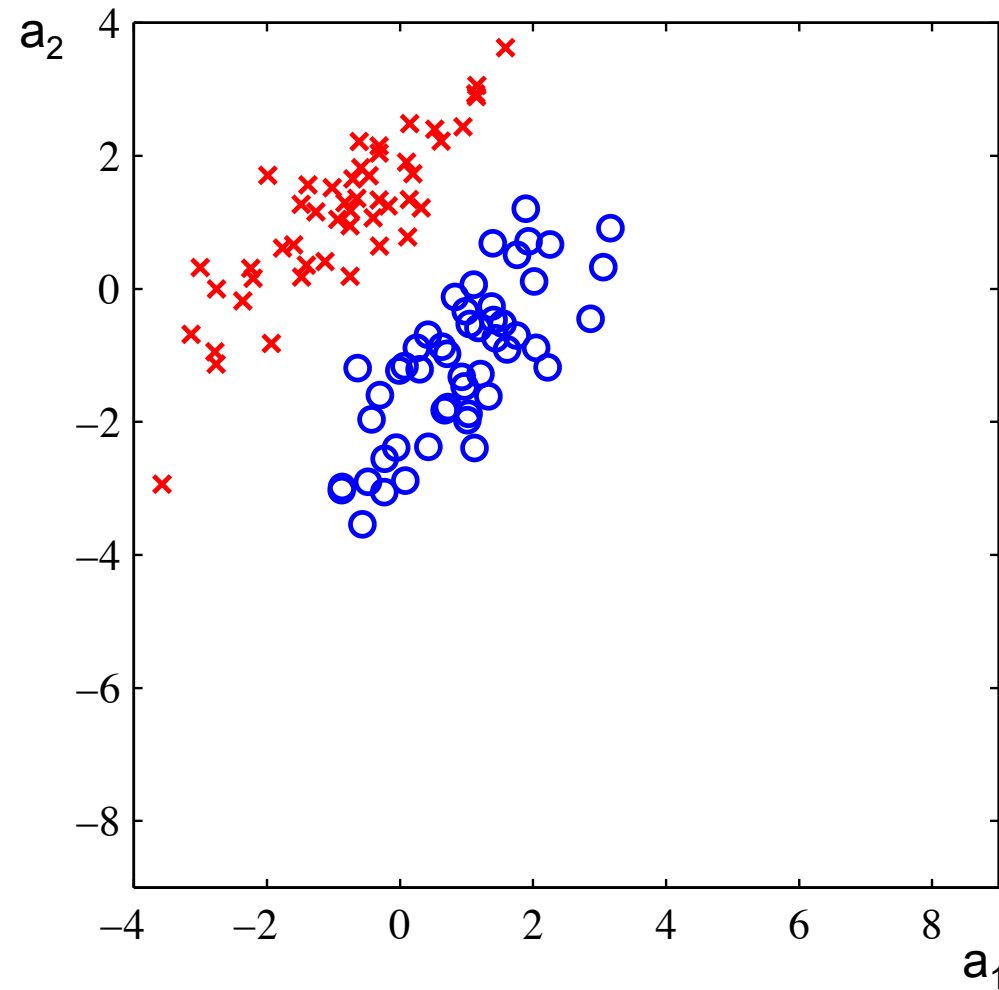
The DATA is a set of tuples $\langle \text{example}, \text{target function values} \rangle$

$$D = \{ \langle \vec{x}_1, f(\vec{x}_1) \rangle, \dots, \langle \vec{x}_m, f(\vec{x}_m) \rangle \}$$

Find a hypothesis h such that...

$$\forall \vec{x}, h(\vec{x}) \approx f(\vec{x})$$

Where would you draw the line?



Linear Discriminants

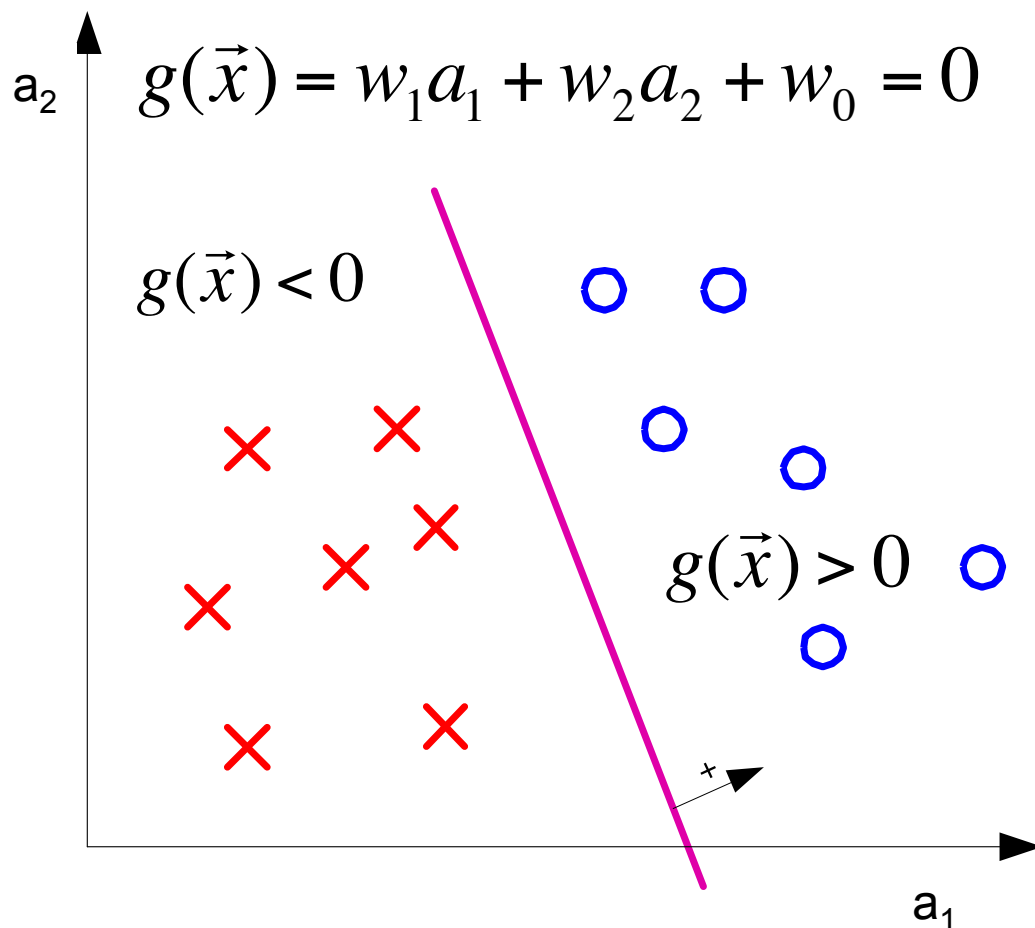
- A linear combination of the attributes.

$$g(\vec{x} \mid \vec{w}, w_0) = w_0 + \vec{w}^T \vec{x} = w_0 + \sum_{i=1}^k w_i a_i$$

- Easily interpretable
- Are optimal when classes are Gaussian and share a covariance matrix

Two-Class Classification

$g(\vec{x}) = 0$ defines a decision boundary that splits the space in two



If a line exists that does this without error, the classes are *linearly separable*

$$h(\vec{x}) = \begin{cases} 1 & \text{if } g(\vec{x}) > 0 \\ -1 & \text{otherwise} \end{cases}$$

Defining a Hyperplane

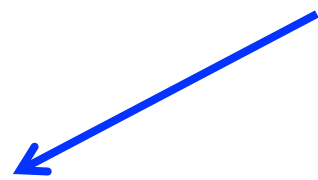
- Any hyperplane can be written as the set of points satisfying the equation below, where \mathbf{w} and \mathbf{x} are vectors in \mathbf{R}^d

$$\vec{w}^T \vec{x} + w_0 = 0$$

- The vector \mathbf{w} is a normal vector: it is perpendicular to the hyperplane. It is often referred to as the *weight* vector. The parameter w_0 determines the offset of the hyperplane from the origin along the normal vector. It is often referred to as the *threshold*.

$$\text{dist to origin} = d_0 = \frac{|w_0|}{\|\vec{w}\|}$$

Euclidean norm



Estimating the model parameters

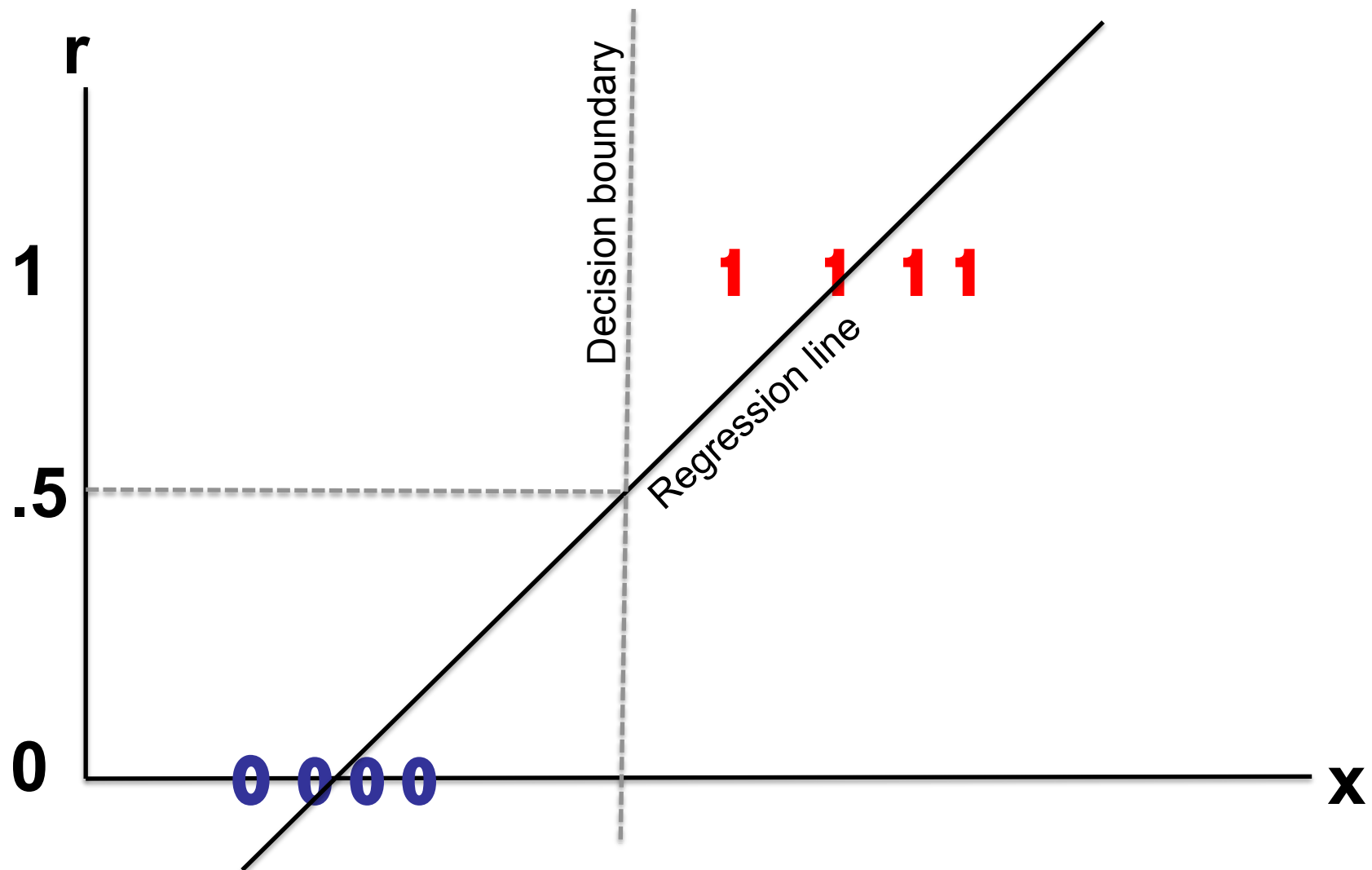
There are many ways to estimate the model parameters:

- Minimize least squares criteria (classification via regression)
- Maximize Fisher linear discriminant criteria
- Minimize perceptron criteria (using numerical optimization techniques, e.g. gradient descent)
- Many other methods for solving for the inequalities using constrained optimization...

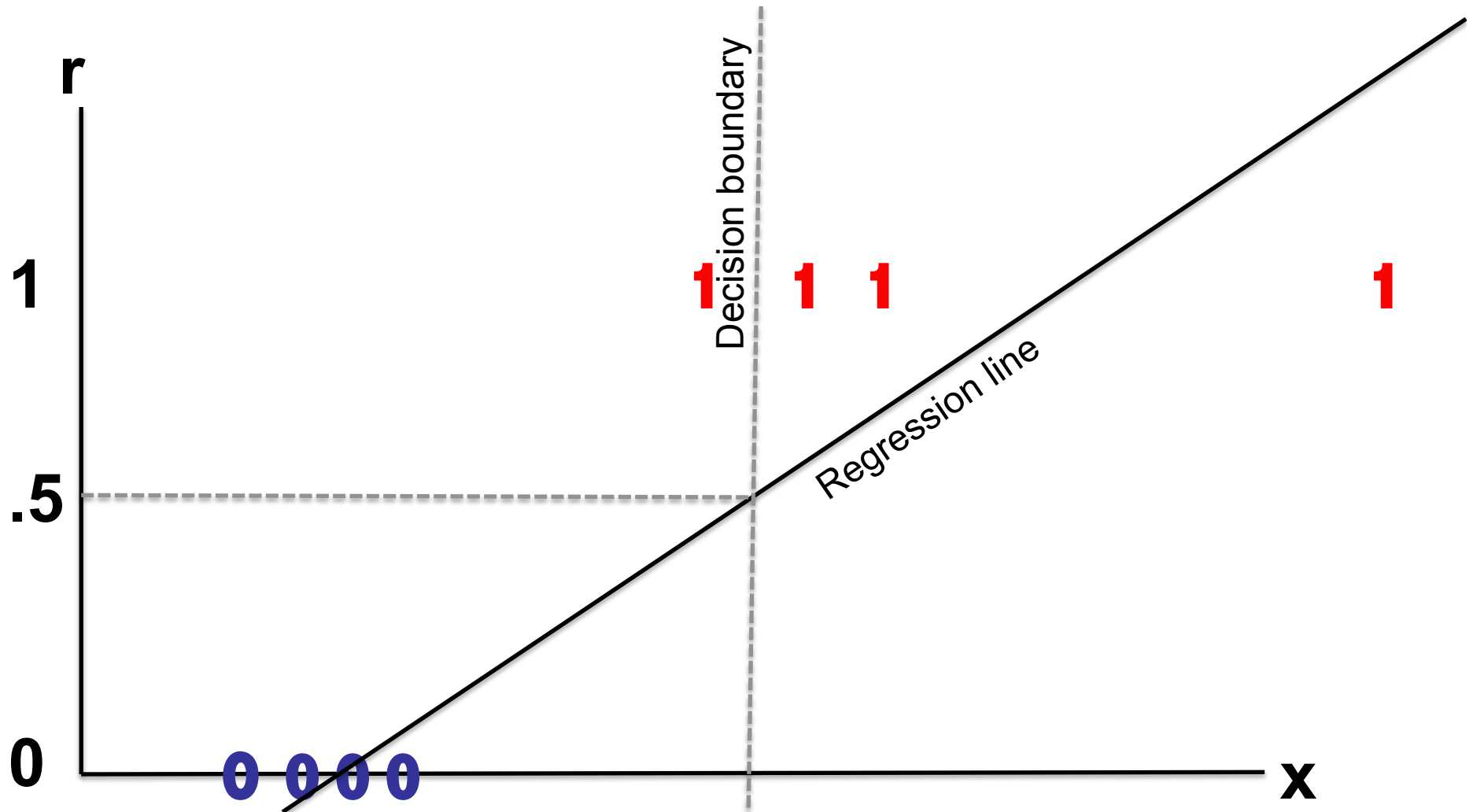
Classification via regression

- Label each class by a number
- Call that the response variable
- Analytically derive a regression line
- Round the regression output to the nearest label number

An example



What happens now?



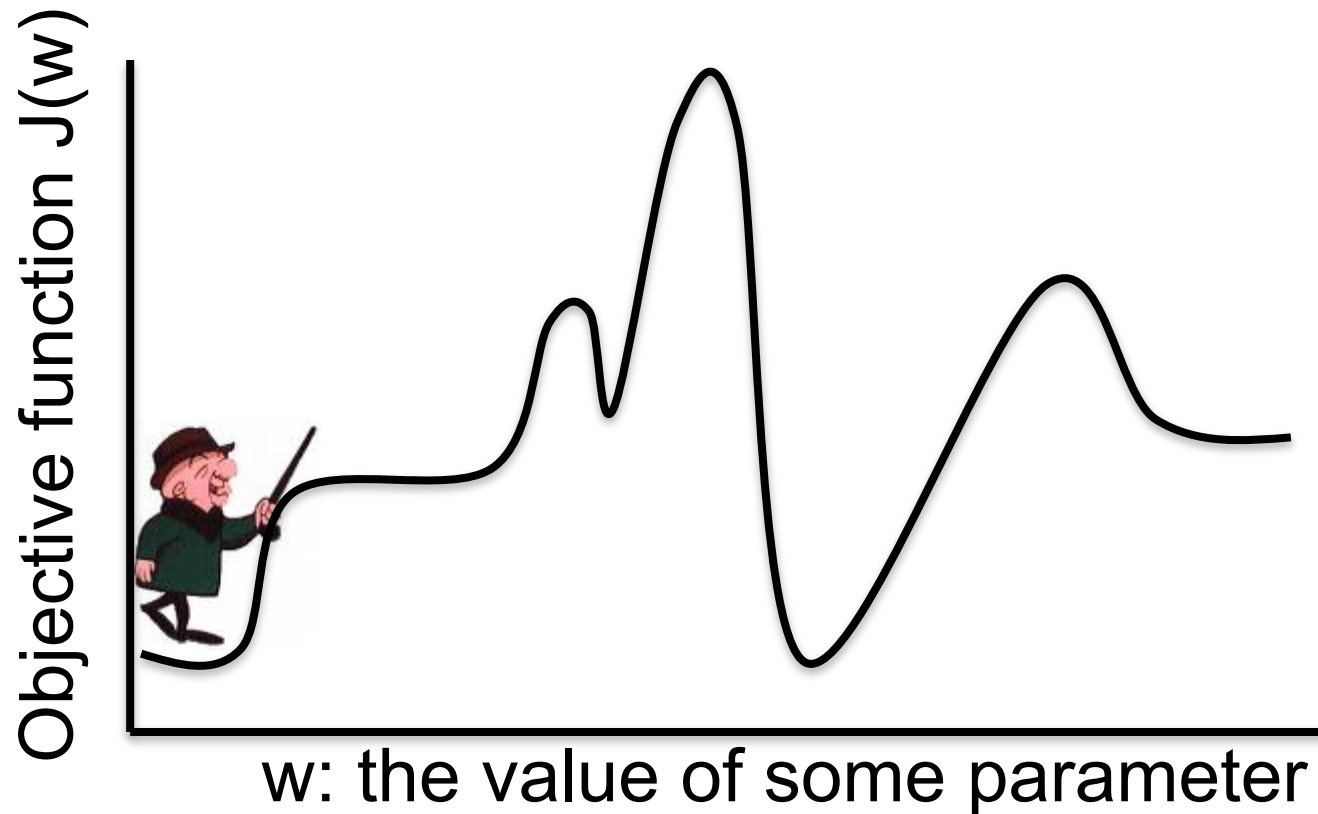
Estimating the model parameters

There are many ways to estimate the model parameters:

- Minimize least squares criteria (i.e. classification via regression – as shown earlier)
- Maximize Fisher linear discriminant criteria
- Minimize perceptron criteria (using numerical optimization techniques, e.g. gradient descent)
- Many other methods for solving for the inequalities using constrained optimization...

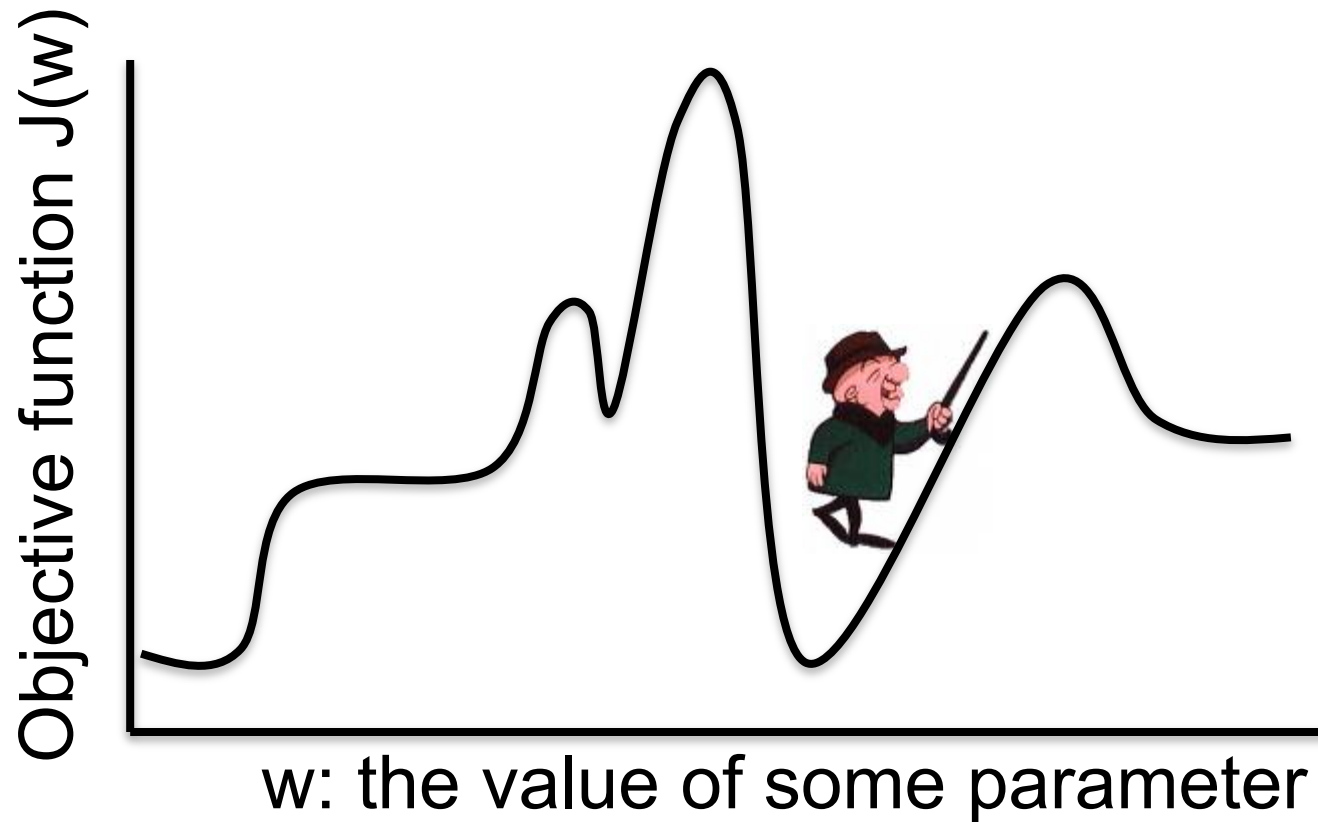
Hill-climbing (aka Gradient Descent)

Start somewhere and head up (or down) hill.

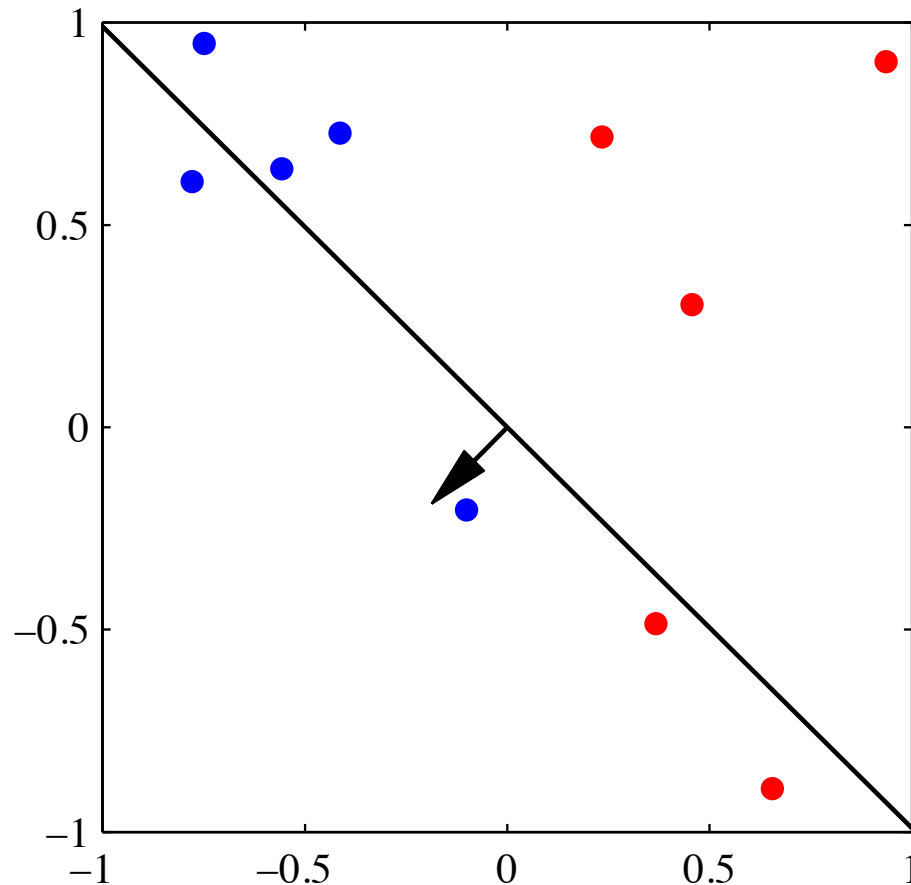


Hill-climbing

Easy to get stuck in local maxima (minima)



What's our objective function?



Minimise number
of misclassifications?

Maximize margin
between classes?

Personal satisfaction?

Gradient Descent

- Simple 1st order numerical optimization method
- Idea: follow the gradient to a minimum
- Finds global minimum when objective function is convex, otherwise local minimum
- Objective function (the function you are minimizing) must be differentiable
- Used when there is no analytical solution to finding minimum

SSE: Our objective function

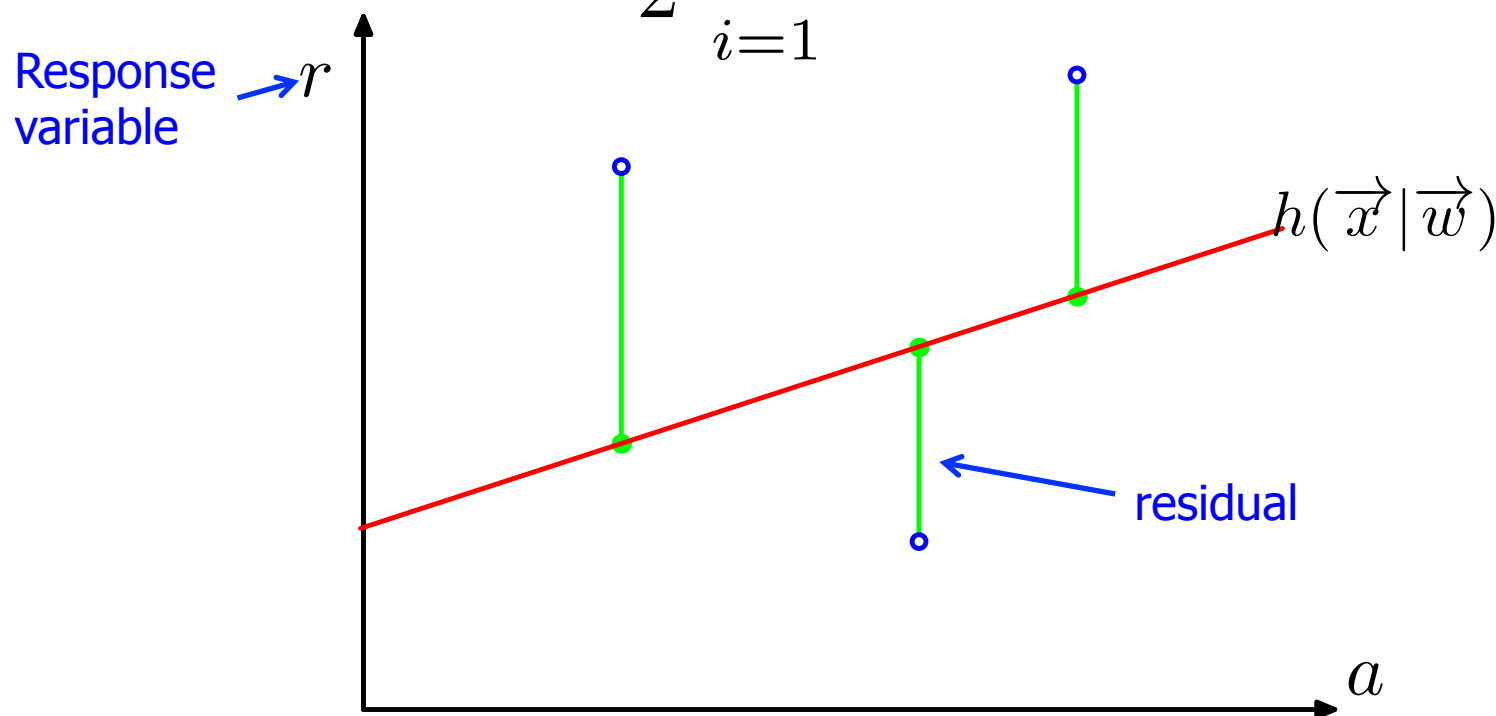
- We're going to use the sum of squared errors (SSE) as our classifier error function.
- As a reminder, we used SSE in linear regression (next slide)
- Note, in linear regression, the line we learn embodies our hypothesis function. Therefore, any change in the line will change our SSE, by some amount.

Simple Linear Regression

Typically estimate parameters by minimizing sum of squared residuals (a.k.a. least squares):

$$SSE = \frac{1}{2} \sum_{i=1}^m [r_i - h(\vec{x}_i | \vec{w})]^2$$

← number of training examples



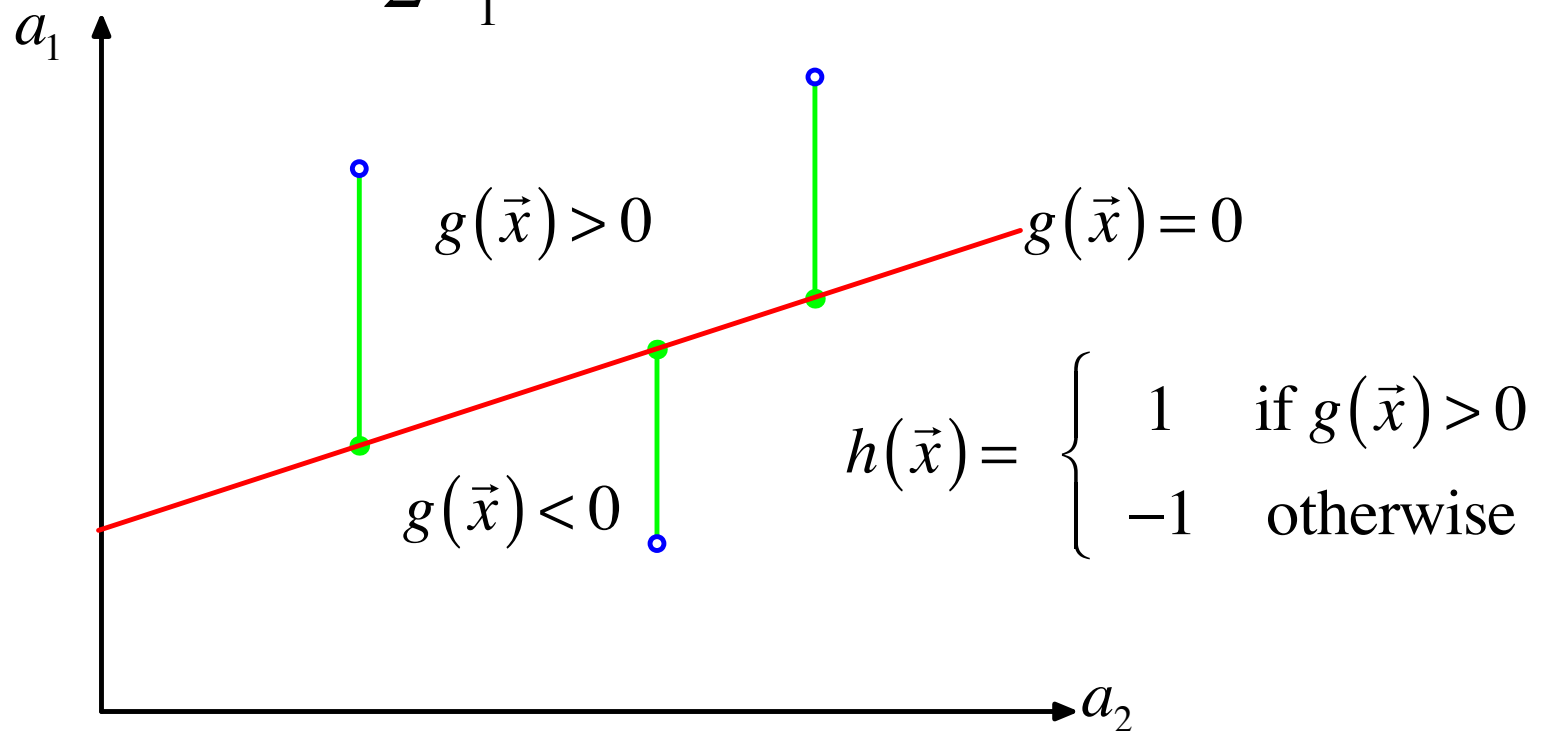
SSE: Our objective function

- In classification (next slide), the line we learn is INPUT to our hypothesis function. Therefore, a change of the line may not change the SSE objective function, if moving it does not change the classification of any training point.
- Note also, in the next slide, both the horizontal and vertical dimensions are INPUTS to the hypothesis function while in the previous slide, the vertical dimension was the OUTPUT. Therefore, the dimension a_1 and a_2 are the elements of the vector X that gets handed to $h(x)$.

Simple Linear Classification

Typically estimate parameters by minimizing sum of squared residuals (a.k.a. least squares):

$$E(\vec{w}) \equiv \frac{1}{2} \sum_1^m (f(\vec{x}_m) - h(\vec{x}_m))^2$$



Gradient Descent

Gradient descent is a useful, simple optimization method, but it can be *very* slow on difficult problems. There are many many optimization methods out there for different types of problems. Take the optimization courses offered in EECS and IEMS to learn more.

Gradient Descent

- \vec{w} are parameters
- θ is the convergence threshold
- η is the step size (in general, important to choose well)
- ∇J is the gradient (vector of partial derivatives with respect to parameters) of the objective function

- Algorithm:

```
begin initialize  $\vec{w}, \theta, \eta(\cdot), k \leftarrow 0$   
    do  $k \leftarrow k + 1$   
         $\vec{w} \leftarrow \vec{w} - \eta(k) \nabla J(\vec{w})$   
    until  $|\eta(k) \nabla J(\vec{w})| < \theta$   
    return  $\vec{w}$   
end
```

Gradient Descent

- In **batch gradient descent**, J is a function of both the parameters and ALL training samples, summing the total error, e.g.

$$\vec{w} \leftarrow \vec{w} - \eta(k) \sum_{i=1}^m \nabla J_i(\vec{w})$$

- In **stochastic gradient descent**, J is a function of the parameters and a different single random training sample at each iteration. – this is a common choice in machine learning when there is a lot of training data, and computing the sum over all samples is expensive.

$$\vec{w} \leftarrow \vec{w} - \eta(k) \nabla J_i(\vec{w})$$

Perceptron Criteria

- Perceptron criteria:

$$J(\vec{w}) = - \sum_{n \in \mathcal{M}} \vec{w}^T \vec{x}_n t_n$$

NOTE: augmented to include the threshold w_0

where $t_n \in \{-1, +1\}$ so that we want $\forall n : \vec{w}^T \vec{x}_n t_n > 0$ and \mathcal{M} is the set of misclassified training examples.

- We can minimize this criteria to solve for our weight vector **w**
- Restricted to 2-class discriminant
- We can use stochastic gradient descent to solve for this
- Only converges when data is linearly separable*

Perceptron Algorithm

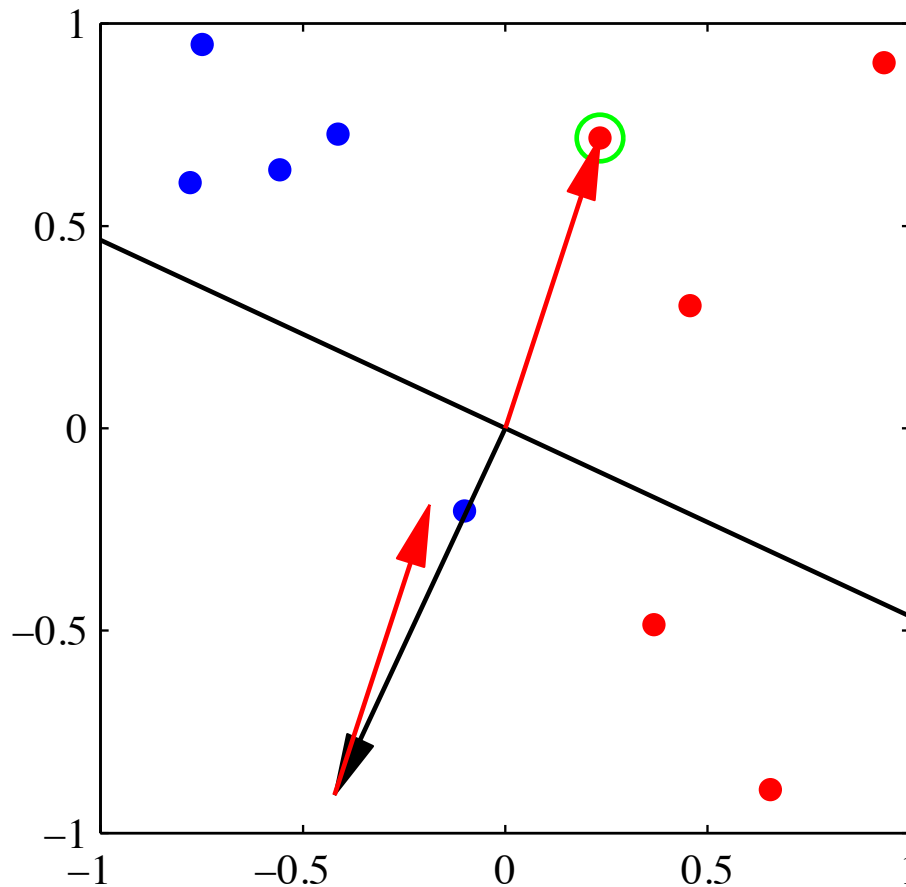
- \vec{w} are parameters
- η is set to 1 (this is fine in this case, since multiplying \mathbf{w} by a scalar doesn't affect the decision)
- $\nabla J_n(\vec{w}) = -\vec{x}_n t_n$

- Algorithm:

```
begin initialize  $\vec{w}, i \leftarrow 0$   
    do  $i \leftarrow i + 1 \bmod m$   
        if  $\vec{x}_i$  is misclassified by  $\vec{w}$   
            then  $\vec{w} \leftarrow \vec{w} + \vec{x}_i t_i$   
    until all examples are properly classified  
    return  $\vec{w}$   
end
```


Perceptron Algorithm

- Example:

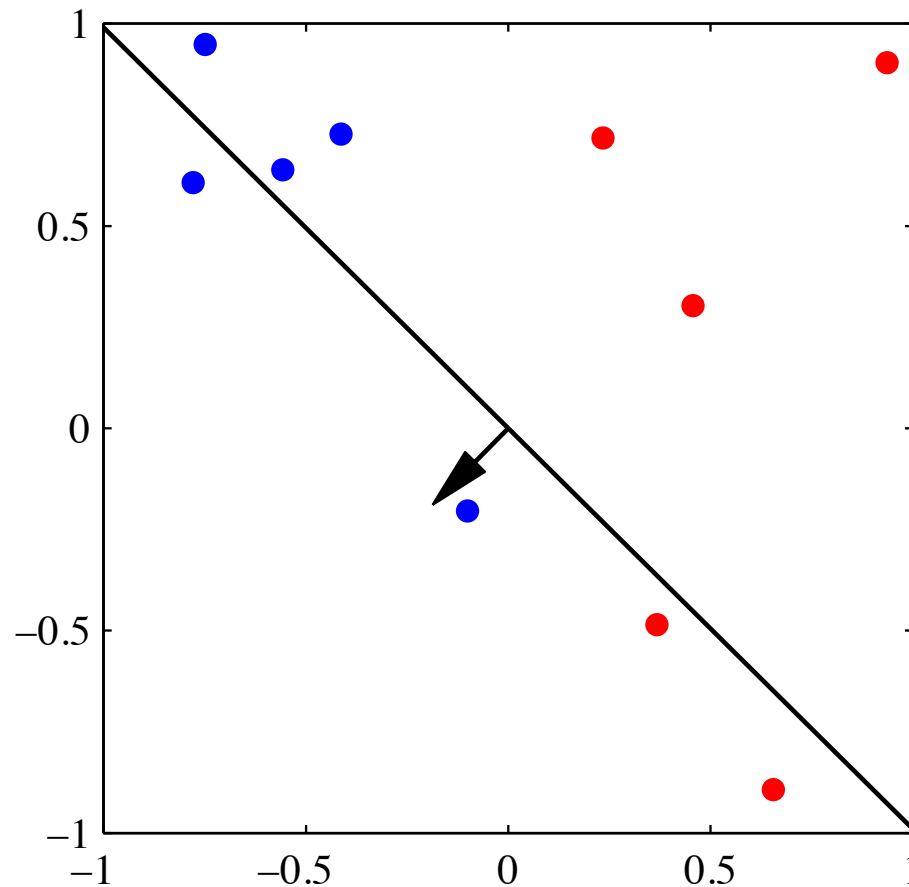


Red is the positive class

Blue is the negative class

Perceptron Algorithm

- Example (cont'd):

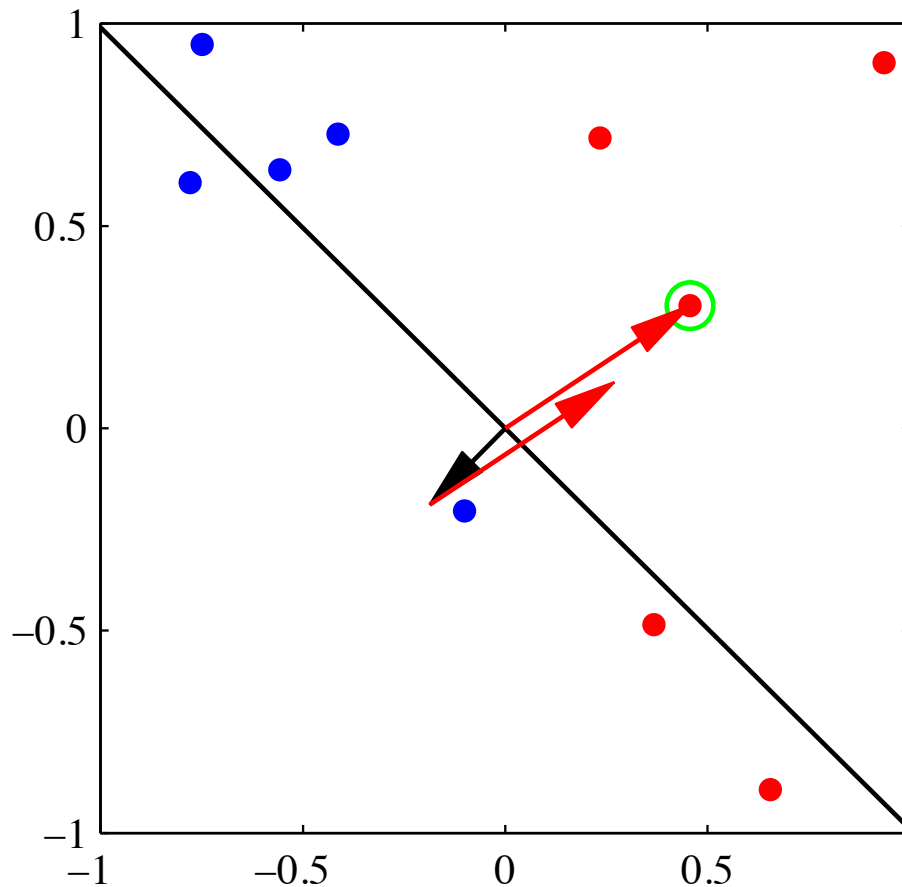


Red is the positive class

Blue is the negative class

Perceptron Algorithm

- Example (cont'd):

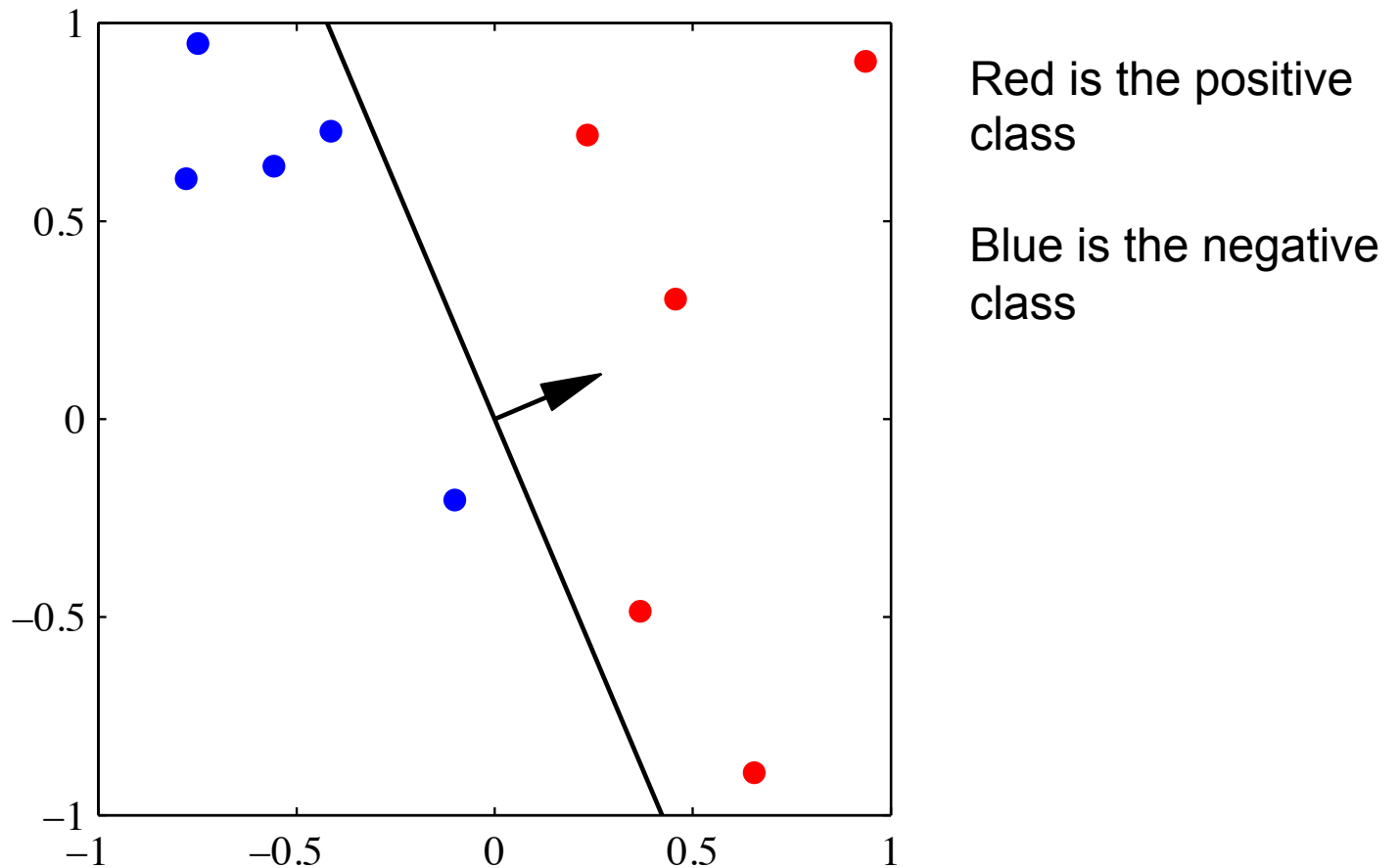


Red is the positive class

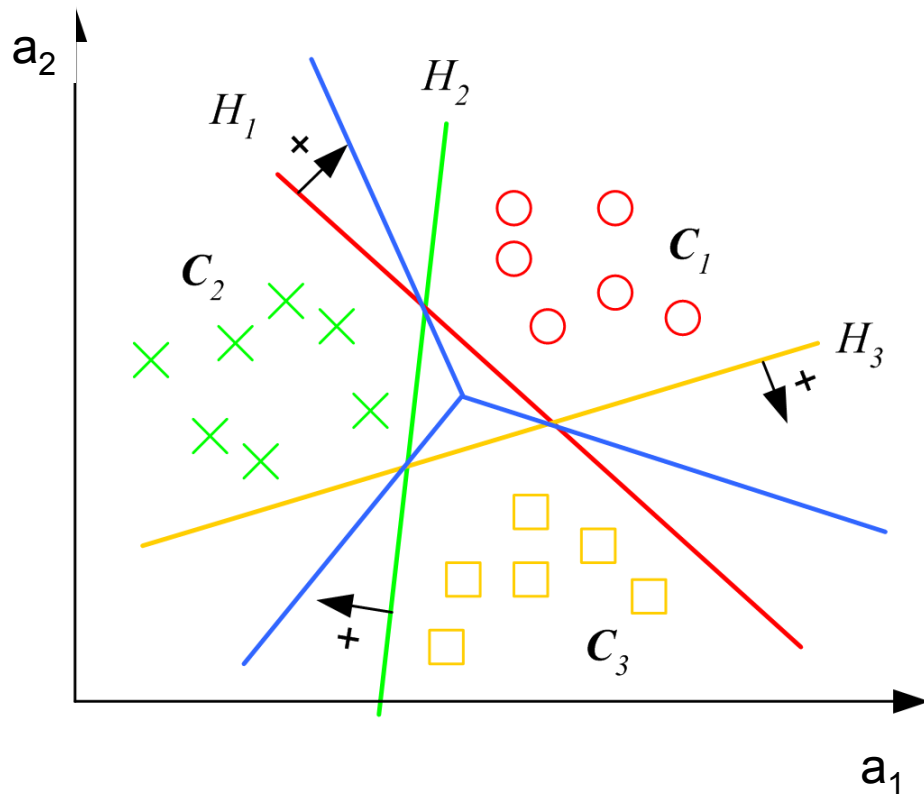
Blue is the negative class

Perceptron Algorithm

- Example (cont'd):



Multi-class Classification

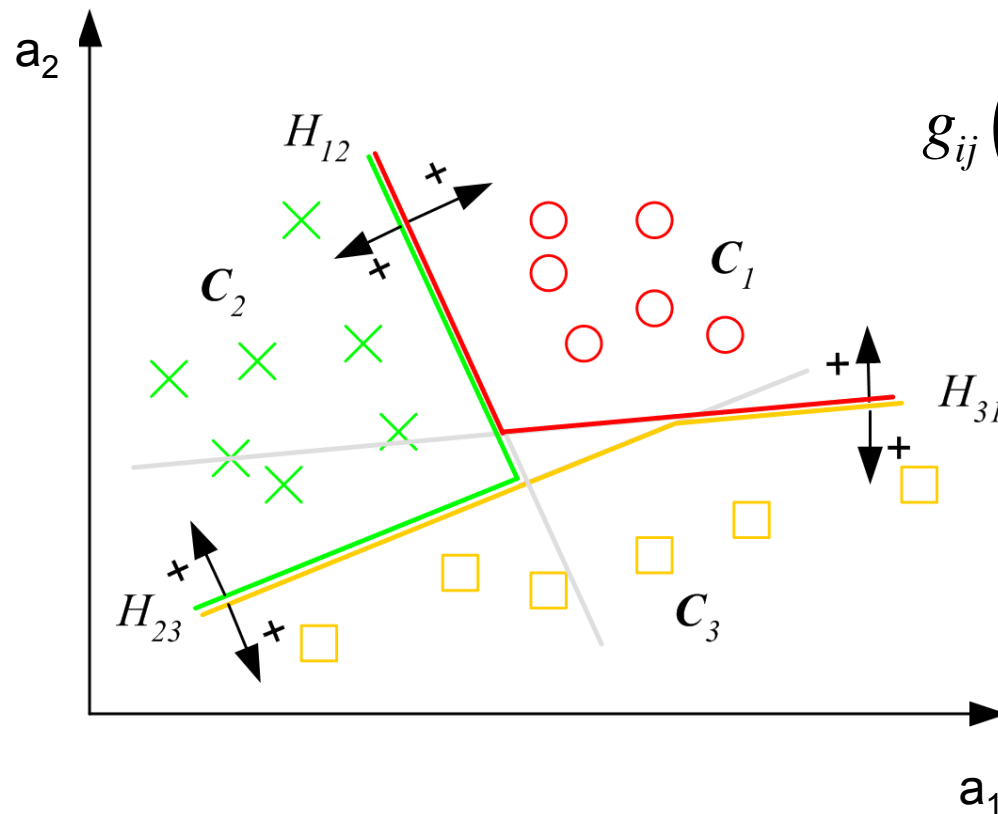


- When there are $N > 2$ classes:
- you can classify using N discriminant functions.
 - Choose the class with the maximum output
 - Geometrically divides feature space into N **convex** decision regions

$$\text{Choose } C_i \text{ if } g_i(\vec{x}) = \max_{j=1}^N g_j(\vec{x})$$

Pairwise Multi-class Classification

If they are not linearly separable (singly connected convex regions), may still be pair-wise separable, using $N(N-1)/2$ linear discriminants.



$$g_{ij}(\vec{x} | \vec{w}_{ij}, w_{ij0}) = w_{ij0} + \sum_{l=1}^K w_{ijl} x_l$$

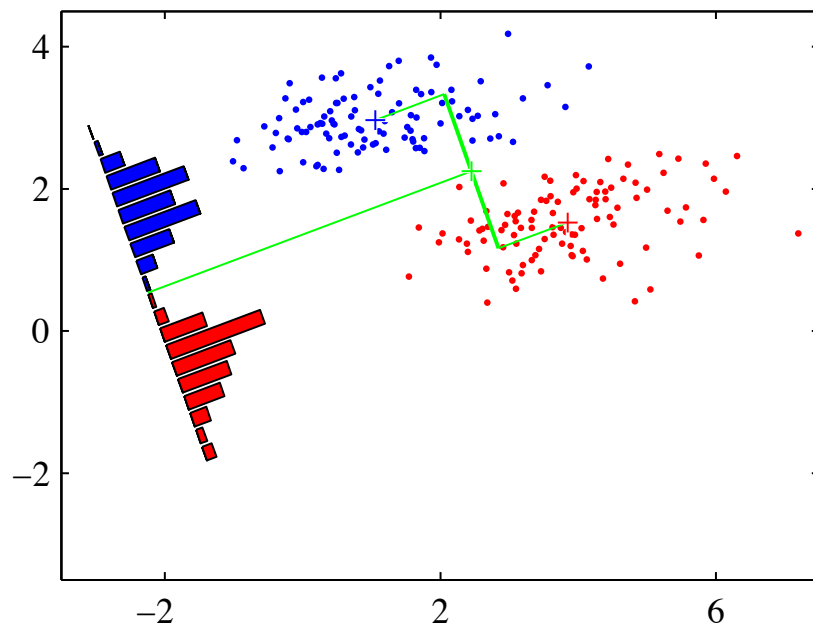
choose C_i if
 $\forall j \neq i, g_{ij}(\mathbf{x}) > 0$

Appendix

(stuff I didn't have time to discuss in class)

Fisher Linear Discriminant Criteria

- Can think of $\vec{w}^T \vec{x}$ as dimensionality reduction from K-dimensions to 1
- Objective:
 - Maximize the difference between class means
 - Minimize the variance within the classes



$$J(\vec{w}) = \frac{(m_2 - m_1)^2}{s_1^2 + s_2^2}$$

where s_i and m_i are the sample variance and mean for class i in the projected dimension. We want to maximize J .

Fisher Linear Discriminant Criteria

- Solution:

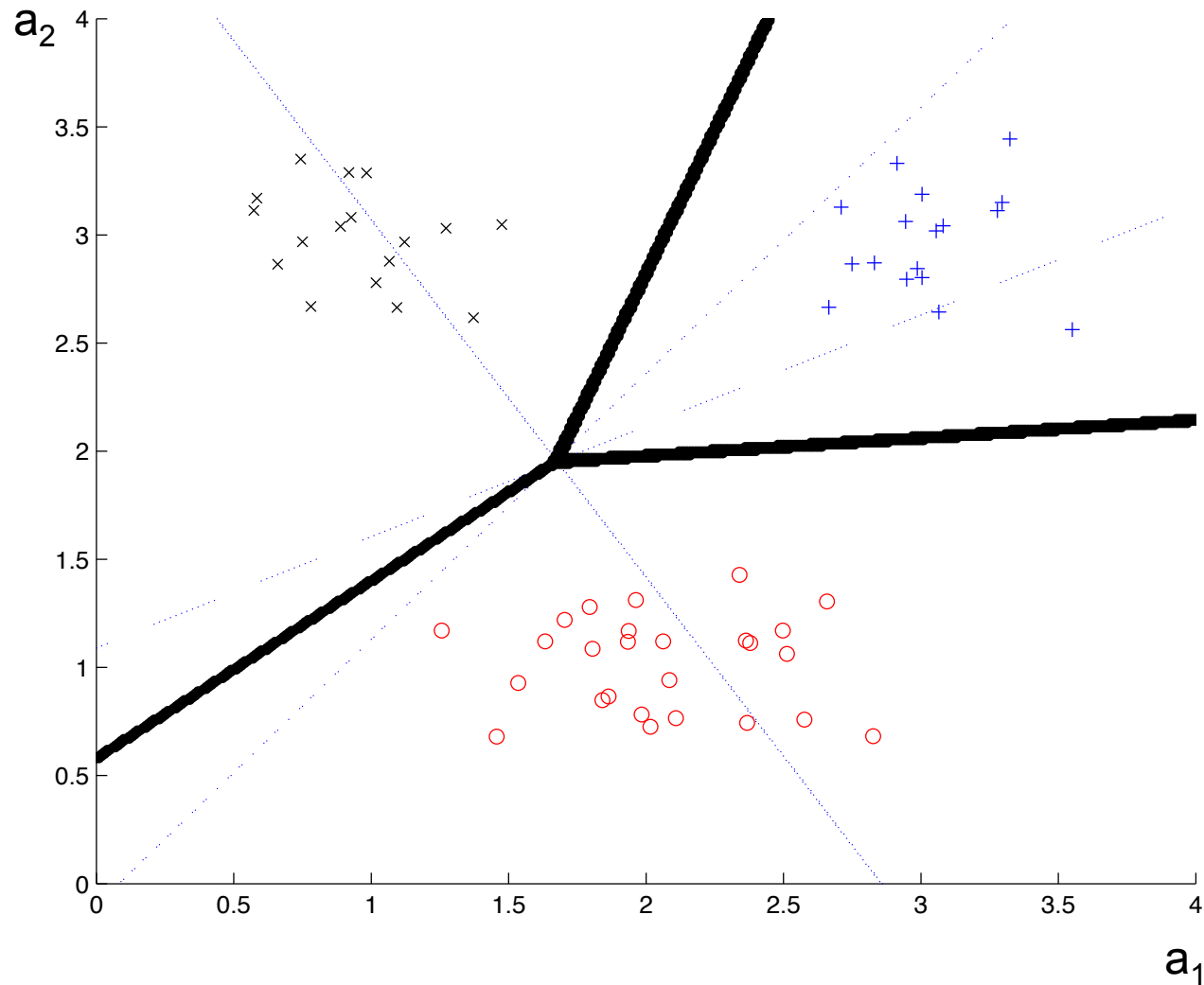
$$\vec{w} = \mathbf{S}_W^{-1}(\vec{m}_2 - \vec{m}_1)$$

where

$$\mathbf{S}_W = \sum_{n \in C_1} (\vec{x}_n - \vec{m}_1)(\vec{x}_n - \vec{m}_1)^T + \sum_{n \in C_2} (\vec{x}_n - \vec{m}_2)(\vec{x}_n - \vec{m}_2)^T$$

- However, while this finds the direction (\vec{w}) of decision boundary. Must still solve for w_0 to find the threshold.
- Can be expanded to multiple classes

Logistic Regression (Discrimination)



Logistic Regression (Discrimination)

- Discriminant model but well-grounded in probability
- Flexible assumptions (exponential family class-conditional densities)
- Differentiable error function (“cross entropy”)
- Works very well when classes are linearly separable

Logistic Regression (Discrimination)

- Probabilistic discriminative model
- Models posterior probability $p(C_1|\vec{x})$
- To see this, let's start with the 2-class formulation:

$$\begin{aligned} p(C_1|x) &= \frac{p(\vec{x}|C_1)p(C_1)}{p(\vec{x}|C_1)p(C_1) + p(\vec{x}|C_2)p(C_2)} \\ &= \frac{1}{1 + \exp\left(-\log \frac{p(\vec{x}|C_1)p(C_1)}{p(\vec{x}|C_2)p(C_2)}\right)} \\ &= \frac{1}{1 + \exp(-\alpha)} \quad \text{logistic sigmoid function} \\ &= \sigma(\alpha) \end{aligned}$$

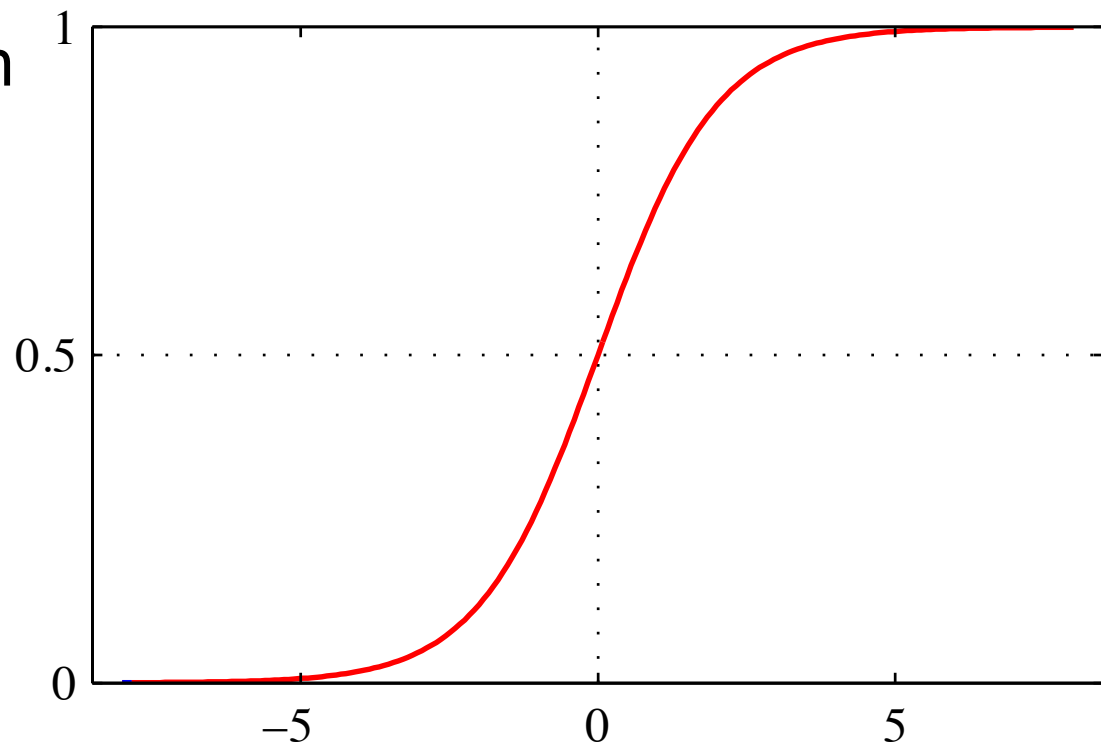
where

$$\alpha = \log \frac{p(\vec{x}|C_1)p(C_1)}{p(\vec{x}|C_2)p(C_2)}$$

Logistic Regression (Discrimination)

logistic sigmoid function

$$\sigma(\alpha) = \frac{1}{1 + \exp(-\alpha)}$$



“Squashing function” that maps $(-\infty, +\infty) \rightarrow (0, 1)$

Logistic Regression (Discrimination)

For exponential family of densities,

$$\alpha = \log \frac{p(\vec{x} | C_1)p(C_1)}{p(\vec{x} | C_2)p(C_2)}$$

is a linear function of \mathbf{x} .

Therefore we can model the posterior probability as a logistic sigmoid acting on a linear function of the attribute vector, and simply solve for the weight vector \mathbf{w} (e.g. treat it as a discriminant model):

$$y = p(C_1 | \vec{x}) = \sigma(w_0 + \sum_{i=1}^k w_i a_i) \qquad p(C_2 | \vec{x}) = 1 - p(C_1 | \vec{x})$$

$$\text{To classify: } h(\vec{x}_i) = \begin{cases} C_1 & y_i > 0.5 \\ C_2 & o.w. \end{cases}$$